OVERVIEW OF ALGORITHMS FOR GRAPH DRAWING

Boštjan Pajntar
Department of Knowledge Technologies
Jozef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Tel: +386 1 4773128
E-mail: bostjan.pajntar@ijs.si

ABSTRACT

This paper presents an overview of many possible types of graph visualization, and also detailed descriptions of the most popular algorithms. Its primal goal is, to help a user, with some relational data on his hands and a need to visualizing it, with the choice of what to use, and what is out there that can be used. However, it can be also viewed as historical overview of graph drawing algorithms, and its evolution.

1 INTRODUCTION

Once, information was a commodity. However, with the development of internet, large amounts of data became publicly available. The problem shifted, from obtaining the information, to extracting the useful one. With, statistical analysis, and even more so, with data mining, we are able to extract the needed information. However, since the tabular view, output of most data mining techniques, is hardly human readable, a need for a visualization arises. Since a picture is worth a thousand words, graph, a way to draw any relational data became a good and very popular way for presenting harvested information.
Quite a few good algorithms for graph drawing were created; some just for certain type of graphs while other for any.

2 OF GRAPHS

Definitions in graph theory vary in the literature. For our needs a loose definition of graph G is a triple (V, E, λ)

- set of vertices V
- set of edges E
- function λ, that maps every end of edge to a vertex

A loop is an edge, which at both ends maps to the same vertex. A multi-edge means two vertices are connected by more than one edge (Figure 1). A graph without loops and multi-edges is simple. If edges are directed, we get directed graph (digraph).

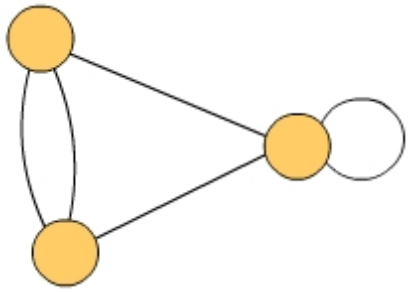

Figure 1: *Graph with one loop and one multi-edge.*

3 GRAPH SIZES

There is no commonly accepted nomenclature for graph sizes. Usually size is defined with the number of vertices, which is not necessarily the best solution. Computers are faster every day, so this numbers varies with time.
Perhaps a better approach is to define the size of a graph with the computational complexity.
We partition graphs to small, medium, large and huge.

- On small graphs one can run more or less any algorithm. This graphs today (2006) are of the order n*10 (up to 150 nodes).
- Medium graphs can be drawn with polynomial algorithms, of low degree. These graphs are also small enough, so they can be drawn on regular screen. Number of nodes n*100.
- Big graphs can no longer be drawn indirectly (we run out of pixels on the computer screen). However, they have to be small enough, to be stored in the memory.
- Even bigger graphs are called huge.

We also distinguish between differently sparse graphs. [5]

- Sparse: $|E| <= |V|$
- Normal: $|V| < |E| <= 3\ |V|$
- Dense: $|E| > 3\ |V|$

This nomenclature considers trees as sparse, hyper-cubes, polyhedra and such as normal graphs.

4 GRAPH EMBBEDING

The quality of graph visualization is of course a subjective judgment. It depends on the graph, and the information we want to extract from it. There are only few studies on how

people read information from graphs [7], most criterions are chosen intuitively.
The prime directive: the user should be able to obtain information from graph, that interest him, as easy as possible.
Some criterions for a good graph representation:

- Vertices should be equally distributed on the screen.
- Small number of edge crossings.
- The length of edges should be uniform.
- The symmetries in the graph structure should be visualized.
- Graf should be bound inside a frame.

These criteria usually well define our demands. However, there are exceptions.

- Planar drawings give better results if edges are of different length (Figure 3, 4)
- For large graphs, clusters of nodes must be used; one of the visualizations is, placing all the nodes in the cluster at one point.
- For very dense graphs, it makes sense to merge edges similar to train tracks. This way it is possible, to obtain a planar drawing of otherwise nonplanar graph. (Figure 2).

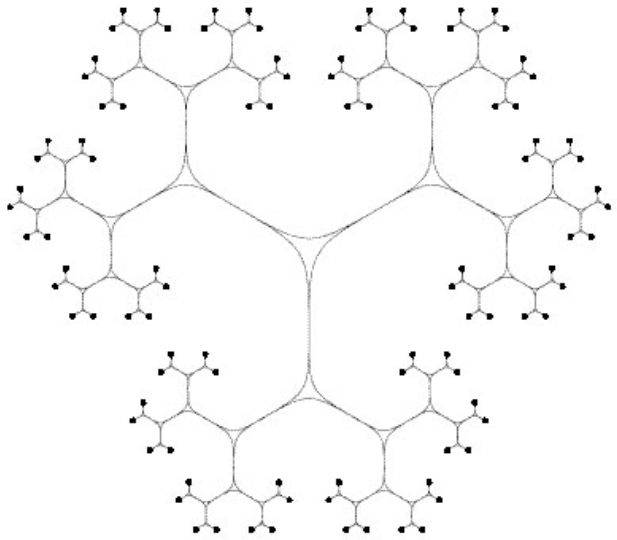

Figure 2: *Full graph on 96 nodes in delta-confluent drawing.*

5 ALGORITHMS

For visualizing graphs many types of algorithms exist. Most common for simple undirected graphs are algorithms that use an analogy from physics. This sort of algorithms will get the most attention in this paper. However, if the graph has some other characteristic, we can employ some other technique.

5.1 Algorithms for graphs with additional properties

5.1.1 Orthogonal style

Graphs can be drawn in orthogonal stile. Usually this method is used for digraphs, as we obtain diagrams. Every edge can bend, but every part is always either horizontal or vertical. This is way edge crossings are always orthogonal. Readability of graph benefits from this.

5.1.2 Sugiyama style

A good way of graph drawing is layered layout. If we have acyclic digraph (DAG), it is possible to layer the graph, on the base of topological numbering. Thus we obtain a flowchart, with all the edges directed in one direction. For example, if there exists an edge from vertex A to B, A is above (or left of) B. The beauty of this approach is that any undirected graph can be converted into DAG. A good algorithm was proposed by Sugiyama [11]. Look also in [8]. The algorithm decomposes into three phases.

1. Distribution of nodes to layers $O(|V| + |E|)$
2. Crossing minimization. (NP-hard, but with good heuristics)
3. Coordinate assignment $O(|E|)$

5.1.3 Planar embedding

Planar graph is a graph, which can be embedded into a plane (or sphere). In other words, there exists an embedding without edge crossings. There is more than one interesting algorithm which achieves this [12]. Any nonplanar graph can be converted to planar, drawn by these algorithms and than previously deleted edges, can be reinserted.

5.2 Drawing on Physical Analogies, historical overview

Here follows overview of algorithms on physical analogies. To every state (drawing) of the graph, we assign an energy value. Smaller energy stands for a better drawing. In this way, search for a good visualization is converted into search of a global minimum of some function. Algorithms vary, in both energy function, and search algorithm.

5.2.1 Spring embedder – Eades [4]

The most intuitive approach to graph drawing comes from a simple physical model. Every vertex is considered a steel ring. Edges between vertices are springs between them. We start with random placing and let the model to be governed by forces. Algorithm stops after a fixed number of iterations. This is a weakness, as different graphs, converge with different speeds.
There are two differences with physical laws. Spring force does not follow Hook law. Secondly forces in algorithm change velocity not acceleration. This difference is

important, as stationary stable state is preferred to a moving one.
Visually the algorithm takes care of two things. Firstly, the edge lengths are uniform, and secondly symmetries in graph structure are visualized. This is a nice surprise, as the algorithm does not aim toward symmetries explicitly.

5.2.2 Kamada and Kawai (KK) [9]

Kamada and Kawai have improved the spring embedder. Algorithm is otherwise similar, except we treat every two vertices as if they were connected by a spring of some length. Spring length is obtained from theoretical length between the vertices, which is the shortest path between them.
Another improvement was that one iteration moves only one vertex. Time complexity in theory remains the same; however, the convergence improves considerably.

5.2.3 Davidson and Harel (DH) [3]

Any energy function can be minimized with simulated annealing (SA). Energy function for the system is chosen, lower energy meaning better result. Move to a worse state is permitted with tolerance to current temperature. With SA the global minima is usually not found; however, the result is usually sufficiently close. One criterion for this algorithm to work is that walleyes of minima are as wide as deep.
Davidson and Harel chose neighborhoods of a state to be any state that is the same as initial with one vertex moved inside a ball of radius r. For better efficiency they chose to decrease r with lower temperature (usually in SA neighborhoods remain the same). For energy function they took a sum of desired properties: repulsion among vertices, frame action on vertices and attraction of edges. At the last stage of fine tuning of SA, when move into bad direction is no longer accepted, they added two more summands. The first one penalized edge crossings, the second one penalizes nodes being to close to edges.
The algorithm is very time consuming, due to SA. In some cases it can produce better results in contrast to others, as more planar drawing can be explicitly asked for. (Figure 3)

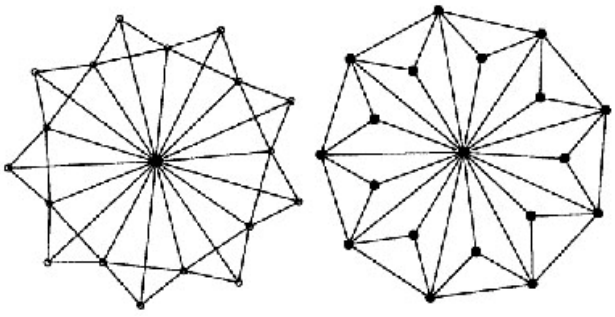

Figure 3: *Same graph drawn with FR and DH*

5.2.4 Fruchterman and Reingold (FR) [6]

Fruchterman and Reingold only demanded 2 criteria for a good graph drawing.

1. Connected nodes should be close.
2. No two different vertices should be to close.

In every iteration we calculate all the attracting forces from connected vertices, and the repulsing forces from all the nodes. Since there is no scalar penalty for edge crossings, the result is a vector, which points out not just how much out of place the vertex is (vector size) but also into which way it should move. The size of actual move is confined to current temperature, which is linearly decreased in every iteration.
This very simple algorithm provides excellent results. Its main advantage is its speed and robustness. Even today, after years of its creation, it is still one of the most popular algorithms for graph drawing.
Due to the fact, we do not penalize edge crossings; the final result often looks as projection of polyhedra into a plane (Figure 4). This can be considered as a feature.

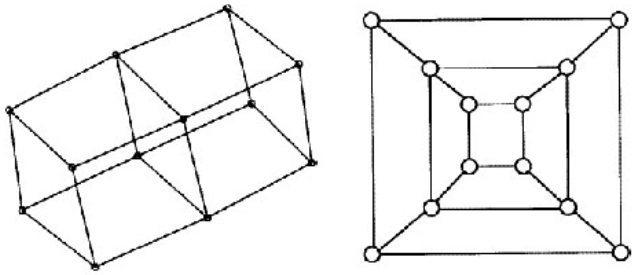

Figure 4: *Graph drawn with FR and the usual plane embedding*

5.2.5 Graph Embedder (GEM) [5]

GEM algorithm improves calculation time of FR. Main improvements are local temperature for every vertex, which provides a way to suppress rotation and oscillations around the optima, and increases a move into the right direction. Secondly gravitation towards barycenter is introduced, which helps disconnected graphs stick together, and also boosts the convergence. Thirdly algorithm is able to stop prematurely if average local temperature is low enough (drawing good enough).
The results are similar to FR algorithm, while the calculation time is better. However, the algorithm is not as robust as FR as some graphs exit algorithm too quickly.

5.2.6 LinLog vertex and edges repulsion [10]

LinLog model was introduced and well accepted at GD03 and GD05 conferences. Three different energy functions are at work; however they can be joined into one model.
One is similar to FR; the other two are created in a way that already tries to cluster graphs. The first of later two is called Vertex-Repulsion LinLog. The results can be seen

in Figure 5. The third energy function, Edge-Repulsion LinLog, is crafted especially for graphs with vertices with a considerable difference of the degree. Nodes with high degree are pushed further apart. This is very useful for social networks, which usually have high degree variance. Apart from different energy function, a Barnes and Hut's algorithm is used for vertex repulsion calculation. Time complexity is reduced from the usual $O(|V|^2)$ to $O(|V|\log(|V|))$.

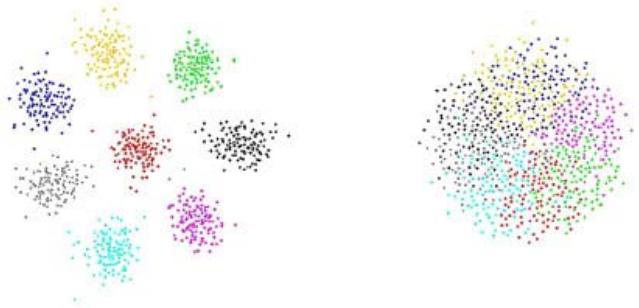

Figure 5: Pseudo-random graph with intra-cluster edge probability 0.16 and inter-cluster edge probability 0.01; left LinLog right FR

5.2.7 Multi-Scale algorithm [13]

If even faster algorithm is required, we can use algorithm that comes from intuition that graph should look nice on all scales. First a decreasing sequence of radiuses r_0, r_1… r_n, is chosen. Than for every radius r_i a locality preserving k-clustering with respect to r_i takes place. Value k is chosen appropriately to the current radius. All the vertices of the cluster are placed to cluster barycenter, and the locale beautification of the clusters takes place.
The algorithm is extremely fast; however, it has problems with graphs of degenerate diameter.

5.2.8 High-Dimensional Embedding [14]

This even faster algorithm runs in linear time! The idea behind it: a good graph drawing in a space of high dimension is easily drawn. Usually 50 dimensions suffice. The second phase is projecting the graph to the usual two or three dimensional space. Algorithm is lightning fast and works perfect on some of the usual graphs i.e. mash, while on more everyday - real life graphs, sometime maps to many points onto a single point. However, placing tens of thousands of nodes in couple of seconds is nice to have.

6 CONCLUSION

What algorithm to choose is up to the user and his needs. Fruchterman-Reingold algorithm is very robust and fast enough to draw most everyday graphs interactively. If we need interactive speed for large graphs, other methods must be used. On the other hand, with larger graphs, the placement is usually not as important as visualization of certain attributes we want to highlight. If graph has some additional data, we can usually harness it.

There are also many "off the shelf" applications under public licenses that provide graph drawing. They are usually written in java and freely available online. The algorithms used, are as of rule the ones presented here, with possible minor improvements.

Acknowledgement
This work was supported by the Slovenian Research Agency and the IST Programme of the EC under PASCAL Network of Excellence (IST-2002-506778).

References

[1] J. Barnes and P. Hut. A Hierarchical O(NlogN) Force Calculation Algorithm. Nature, 1986.
[2] Dirk Beyer and Andreas Noack. Clustering software artifacts based on frequent common changes. In IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension, pages 259–268, Washington, DC, USA, 2005. IEEE Computer Society.
[3] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. ACM Transactions on Graphics, 15(4):301–331, 1996.
[4] Peter Eades. A heuristic for graph drawing. 42:149–160, 1984.
[5] Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs. In Roberto Tamassia and Ioannis G. Tollis, editors, Proc. DIMACS Int. Work. Graph Drawing, GD, number 894, pages 388–403, Berlin, Germany, 10–12 1994. Springer-Verlag.
[6] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by forcedirected placement. Softw. Pract. Exper., 1991.
[7] Weidong Huang and Peter Eades. How people read graphs. In APVIS, pages 51–58, 2005.
[8] Michael J¨unger and Petra Mutzel, editors. Graph Drawing Software. Springer Mathematics and Visualization Series, 2004.
[9] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. Inf. Process. Lett., 1989.
[10] A. Noack. An energy model for visual graph clustering, 2003.
[11] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. SMC-11(2):109–125, February 1981.
[12] W. T. Tutte. How to draw a graph. In Proceedings London Math. Society, pages 743–768, 1963. Graph drawing.
[13] D. Harel, Y. Koren. A Fast Multi-Scale Method for Drawing Large Graphs, Journal of Graph Algorithms and Applications, vol. 6, no. 3, pp 179-202, 2002
[14] D. Harel, Y. Koren. Graph Drawing by High-Dimensional Embedding. Graph Drawing: 10t International Symposium, GD2002, Irvine, CA, USA, August 26-28, 2002